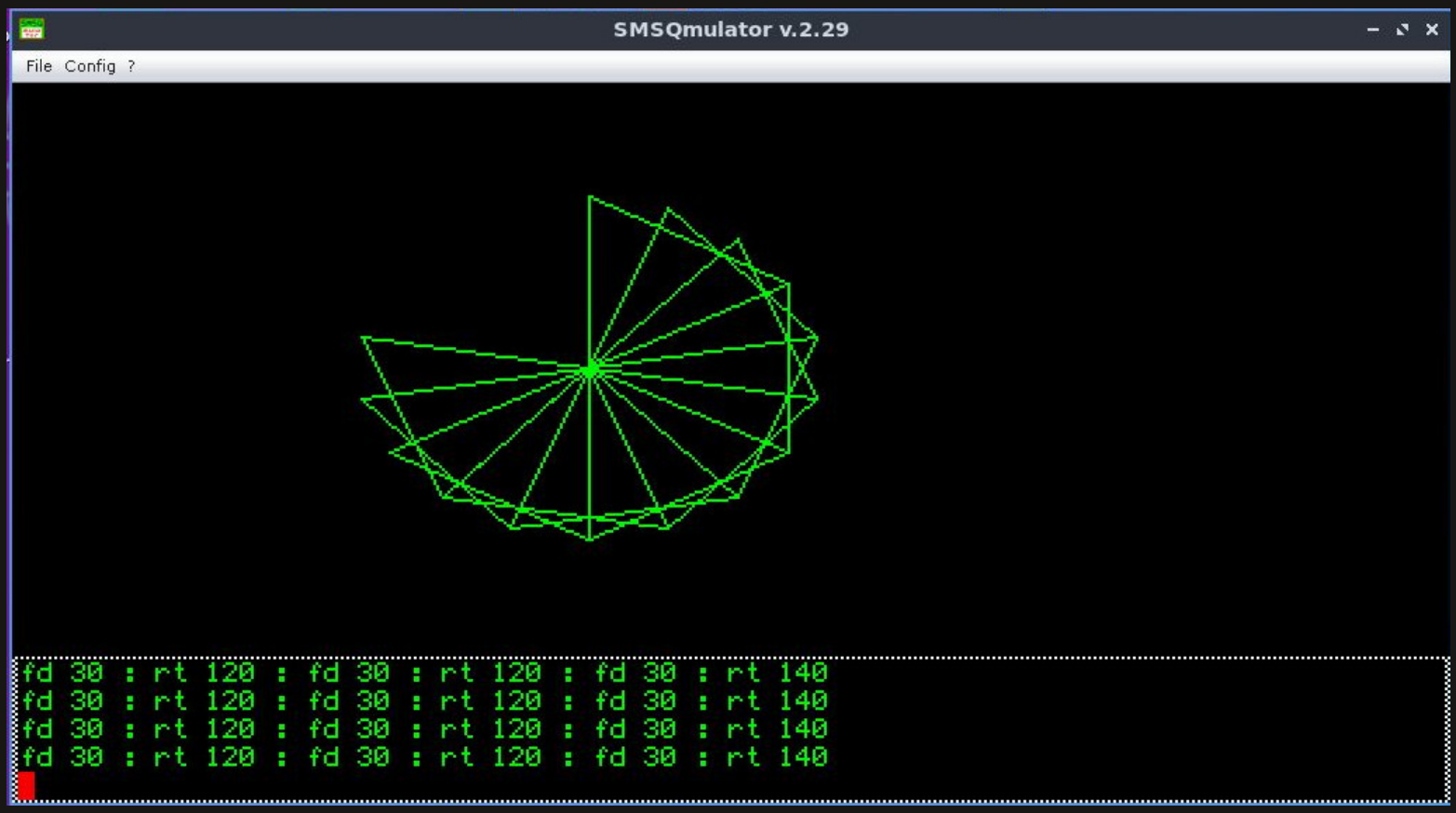
SMSQzine

Issue #8 **Oct. 2021**



SMSQzine table of contents

Published by:
Timothy Swenson
swenson_t@sbcglobal.net
swensont@lanset.com

SMSQzine is published as a service to the Sinclair QL community. Writers are invited to submit articles for publication. Readers are invited to submit article ideas.

Created using Open Source Tools:

- OpenOffice
- Scribus
- Gimp
- SMSQmulator

Editorial

It has been a while since the last issue of SMSQzine. I have been working on a few things, but work and laziness has kept me from finishing new articles.

A couple of articles in this issue were written over a year ago, so they might seem a little out of date, but I wanted to put them in for those that might not be regular readers of the forum.

As I am working to finish this issue, news was released on the death of Sir Clive Sinclair. I don't know about the Sinclair users in the UK, but those of us in the US, Sir Clive is really know for the computers and other devices that carried his name. We knew him as someone that was shooting for "bang for the buck". His computers were always the low price on the market. It cause some grief from the computer press, and not all buyers like his computers, but for a number of us in the community, we liked his goal of low price with decent performance.

I bought my ZX81 mostly due to the price. The ZX81 was $250 for B&W and 16K, where as the Vic-20 was color and only 5K for $400. The Tandy Coco was color and 16K for $600. The Apple II was selling for $1,200. At the same time I bought a 10-year old used car for $600. I was really looking for bang for the buck.

Despite Sinclair being out of the main US market in 1984, I kept up with Sinclair and bought the QL in April 1986. Later I bought a Z88. Ever single computer has its limitations, but I found them all usable with something to learn.

As the ZX81 reaches it's 40th birthday and the QL its 37th birthday, I still find tinkering with the enjoyable. Thank you Sir Clive.

Update to sQLux

Graham has recently updated sQLux. sQLux is uQLx but built using the SDL2 graphics framework. For those looking for a QL/QDOS emulator for Linux, sQLux is your best option. It is doubtful that the fixes for sQLux will be back ported to uQLx. SDL2 is required for sQLux, but that is a fairly simple install.

A number of key fixes are:
- TCP/IP fix to support QLirc2.
- Adding a sqlux.ini file that can supersede the .uqlxrc file.
- Removal of some older code that was not needed.
- Some general stability fixes.

Most of the changes were made 5 months ago, but a few have been made within the last two months.

sQLux can be downloaded as a zip file from Github:

https://github.com/SinclairQL/sQLux

Once downloaded and unziped to a directory, just run cmake and then make to compile the program. cmake is new and you might need to install it. The binary is called sqlux.

QLirc

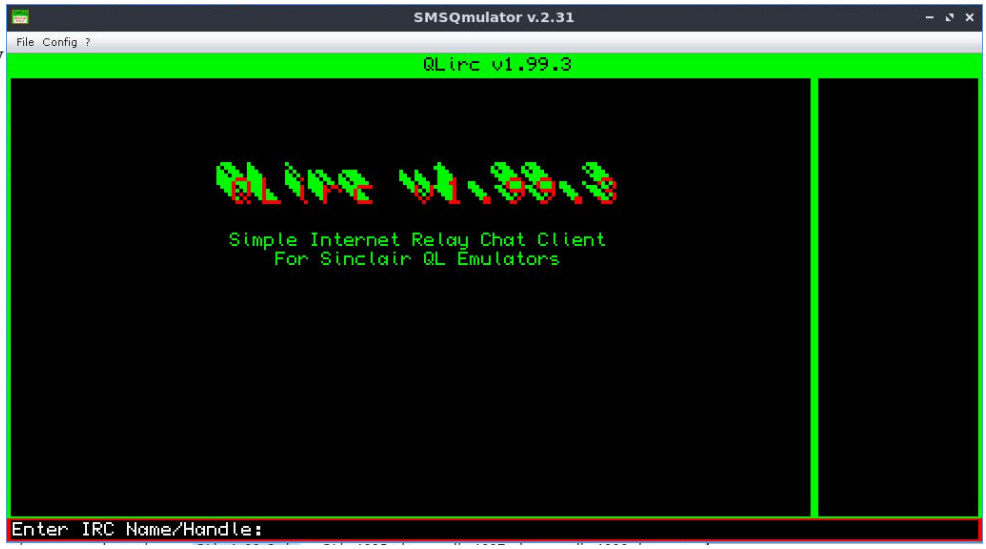
I have been active on the QLForums chat for over 6 years. I know how important IRC chat can be for communication with other QLers.

A while back I was thinking about projects for the QL and was thinking about a network-based project. I know that others have been working on the idea of setting up network communications between QL emulators. That's not something I was interested in doing.

Dilwyn Jones wrote a short demo program showing how one could download any web page in SuperBasic. I thought about a web browser, but they can be very complicated.

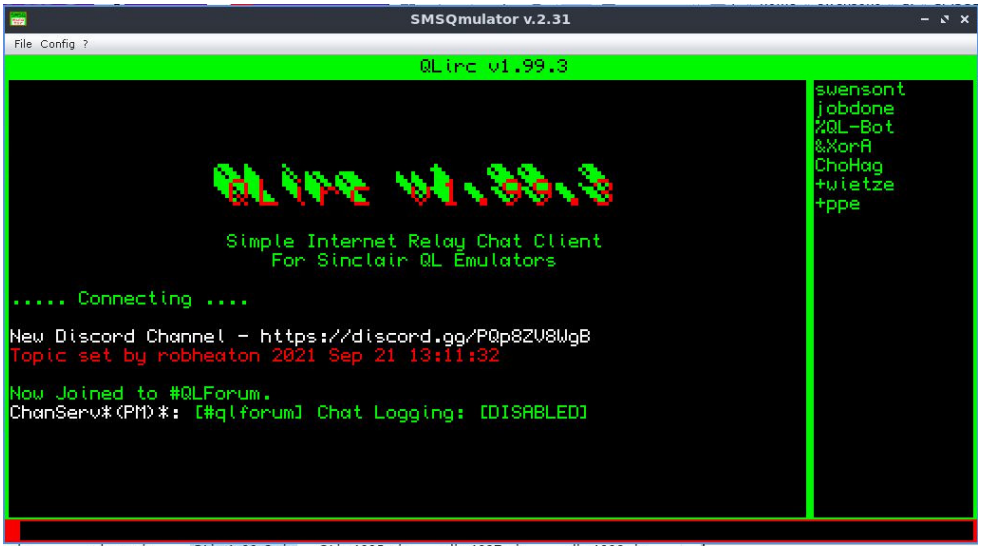
Then it struck me to write a IRC chat client for the QL. The idea was to create a very simple client just to see if it could be done. I first had to learn more about IRC and how the protocol works. I used Wireshark (a packet analyzer) to see how the Pigin

IRC chat client worked and how it established a connection to the chat server.



I then started sending and receiving command strings with the IRC chat server from SuperBasic. The program would send a string, then read a string, in the normal ping-pong style that most network protocols have.

I found a bug in SMSQmulator that caused the emulator to lock up when getting input from the network. Wolfgang was able to quickly fix that issue for me and I continued on my work.



In the end I came up with a chat client, QLirc, that would log into a chat server. It would get a list of users logged into chat and display them. It could send and receive chat messages and handle a little bit of the IRC protocol, just the bare minimum.

To make sure that the client was not stuck waiting for user input and ignoring input from the network, I set up some polling where the client could poll the user for a character, then poll the network for a character, and so on. If there was no user input, the program would skip it and check for network input.

When done, I had a demonstration version that could be used for connecting to the QL forum chat server and get online. The program worked in SMSQmulator and QPCII.

I kept the project quiet as I did not want to get a bunch of folks interested in case I could not get it to work. Those that were on chat late my time (very late UK time) could see my testing.

When announced to the QLforum chat, a number of folks were interested. Surprisingly, Dilwyn Jones took a large interest in the program and wanted to expand it from beyond the demo version. Granted the demo version was usable to get onto chat, but Dilwyn wanted to go further.

Taking my code as a starting point, Dilwyn spent hours studying up on the IRC protocol and started expanding the program. I wrote the demo in Structured SuperBasic (my tool that allows SuperBasic to not have line numbers) and I was able to convince Dilwyn to continue using SBB for this project.

In the end, Dilywn has produced QLirc2, a much better IRC chat client that is quite capable. It is more functional. It supports the IRC / commands, such as "/name" to change a nickname. The client supports more than just the QL forums. The default is the

forums, but it can be modified to connect to other IRC servers.

For the casual viewer, QLirc1 and QLirc2 may look alike, but there there is far more with QLirc2. The source code is much bigger than QLirc1. QLirc1 is 4K of source code. QLirc2 is 71K of source code. QLirc2 is much more capable and more bullet proof.

With his testing, Dilwyn has found a few issues with Q-emulator and UQLX and sQLux. Danielle has fixed the issue in Q-emulator. Graham has fixed the issue in sQLux. So, the four emulators that support TCP/IP will work with QLirc2. I personally have tested QLirc2 with sQLux and SMSQmulator, both running at the same time on the same computer.

It is hoped that other QLers will test out QLirc2 and connect to the QL Forums chat. The most recent released version of QLirc2 will be part of the zip file that comes with this issue.

Prime Number Benchmark

In the January 1990 issue of the Capital Area Timex Sinclair (CATS) Users Group newsletter, Duane Parker published a prime number program that will generate prime numbers from 32767 to X. The program was used as a way to test the speed differences between languages and compilers on the QL. Duane provided examples in Forth, Pascal and SuperBasic. A few months later Herb Schaaf provided an example in C with the Small-C compiler. A while after that I ported the program to BBC Basic for the Z88.

Recently I was digging thought some CATS newsletters and ran across the benchmark and thought that I would try it again on the QL using SMSQmulator and sQLux. I modified Herb's original Small-C and wrote versions for Digital Precision C, GST QC and C68. Running the benchmark from 32667 to 1000 (the original value) was way to fast time. It ran in under a second. I them modified the program to run through the benchmark 50 times. In all of the versions, the results of the benchmark were printed to the screen, which is supposed to slow processing down. The results of running the benchmark are:

SMSQmulator:

DP C	14 sec
QC	12 sec
C68	20 sec
Basic	54 sec
Turbo	46 sec

sQLux:

DP C	14 sec
QC	13 sec
C68	19 sec
Basic	265 sec
Turbo	32 sec

The biggest thing to note is how slow SuperBasic was on sQLux compared to SuperBasic on SMSQmulator. This is really the different between QDOS and SMSQ/E. For QDOS, using Turbo to compile your program saves processing time. On SMSQ/E, the SuperBasic interpreter runs faster and compiling with a SuperBasic compiler is not really needed.

Another interesting take away is that both DP C and QC both are faster the C68. If you really need speed and your program is fairly simple, then DP C and QC are viable candidates. As for executable size, DPC was the smallest at 3K, QC at 8K and C68 at 24K.

Documentation

I love good documentation. I like good looking documentation. The QL is not a good platform for making good looking documentation, such as a PDF file. For a while I have been taking QL documentation and making them look better. I'll take text files or Quill documents (which are converted to text files) and load then into Open Office. From there I make them look better.

My next step is to print them out so that I have a hard copy to read and keep nearby for reference. I also keep electronic copies on my different systems they are easier to carry that way.

After having done as a number of documents, I have decided to put them on my web page so others can use them. The list of documents is:

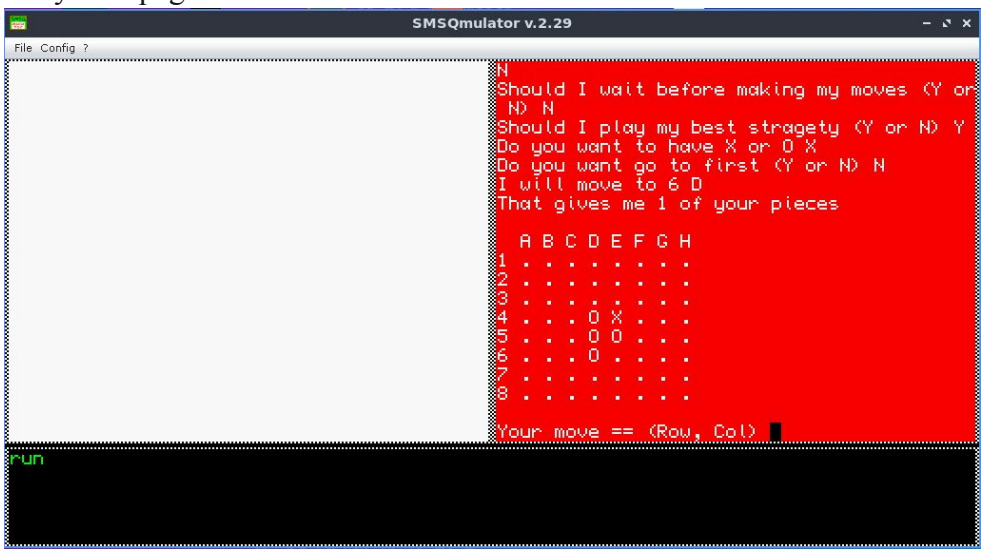
C68 & PE Tutorial
C68 Documenation
C68 Library
Small-C compiler
Small-C Compiler Companion
Digital Precision C compiler
GST QC C Compiler
Structured SuperBasic
Z80ASM cross assembler
QED
QED2
Xtricator ZX81 emulator
Xchange
  Abacus
  Archive
  Easel
  Quill
  Xchange
  Import/Export

All of these are on my website for download.

Early Othello Programs

I've been looking into early versions of Othello programs. There reference to a program written in BCPL by N. J. D. Jacobs in 1977. I am unable to find a copy of the source code for that program. In the November 1977 issue of Creative Computing was another Othello program by Ed Wright. The program is in FORTRAN but I'm too rusty on my FORTRAN to see if it will work with the FORTRAN compiler on the QL.

In the October 1977 issue of Byte magazine, the article "Othello, A New Ancient Game", Richard Duda published a BASIC version of Othello. There is little discussion in the article about how the games plays, other than the two strategies:



1. Take the maximum number of pieces
2. Add a bonus for outside position

The game is designed to run on a video terminal or a teletype. The program does not print just the changes to the board, but prints the entire board after each move.

The article does not say what version of BASIC it is written for. There is little in the syntax used that would indicate which version of BASIC. It could be DEC BASIC or a form of MicroSoft BASIC. The program only had a few, what I could call non-standard, commands that needed to be modifed to run on the QL.

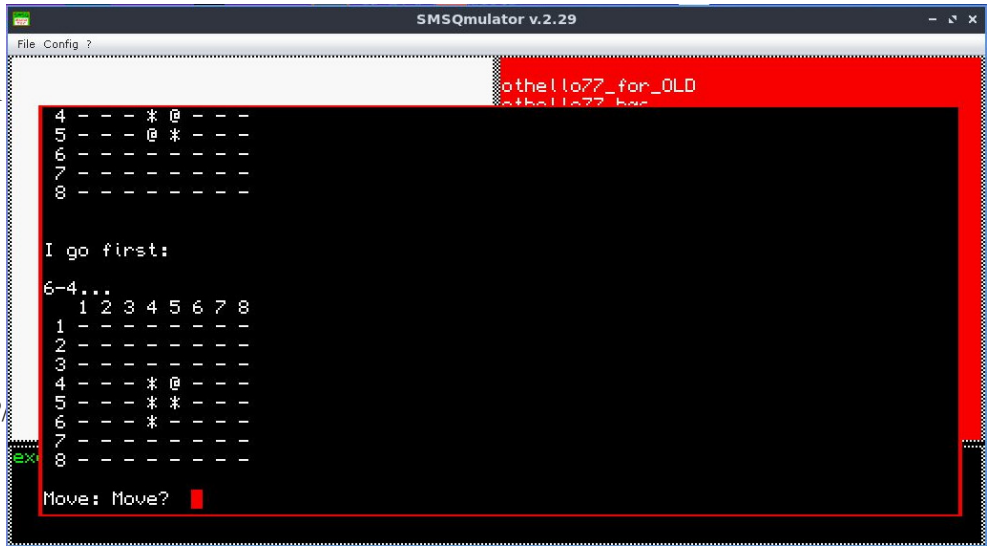
It has the statement like "DIM A(9,9), I4(8),..." where a single DIM is used to set up a numbers of arrays. Obviously this was easy to fix. The random number, RND, command is usually different between systems, so that had to be change. The other oddity was not using GOTO after a THEN statement. The program would have:

```
IF x > 0 THEN 250
```

For the QL, it has to be changed to:

```
IF x > 0 THEN GOTO 250
```

Another version of Othello I came across was written in C by Robert Halstead at MIT. Leor Zolman wanted to run the program on on his own computer, but there was no C compiler for CP/M, so he wrote one. The version of Othello was written some time before 1979.



LOGO, but more of LOGO like wrapper to the turtle graphics commands built into the QL. The term "turtle" graphics actually comes from LOGO. It was the "turtle" that moved on the screen, drawing lines behind it.

I was able to find the BDS C version online. There was the original version and an ANSI version. The original version called qsort which had some issues, so the ANSI version used its own bsort. It was this version that I was able to compile with C68.

Both versions did use kbhit() to detect keyboard input. That comes with conio.h and C68 did not have that header file. The function was only used in demo mode, so I just commented out that section and let it compile.

Like the first version, BDS C Othello prints out the while board after each turn. After you enter your move, the computer comes up with it's move and then prints out the board. It does take a bit of getting used to. Instructions for the game are in the header to the source code.

Both programs are small enough to see how the a simple version Othello is done.

QL Logo

A recent discussion on QL forum chat reminded me of a version of LOGO that I wrote for the QL some time ago. It is not a full proper implementation of

The program is not really something that you run, but a collection of SuperBASIC procedure that are similar to LOGO commands. Once the program is loaded, two commands must be run to start it all:

```
init
cs
```

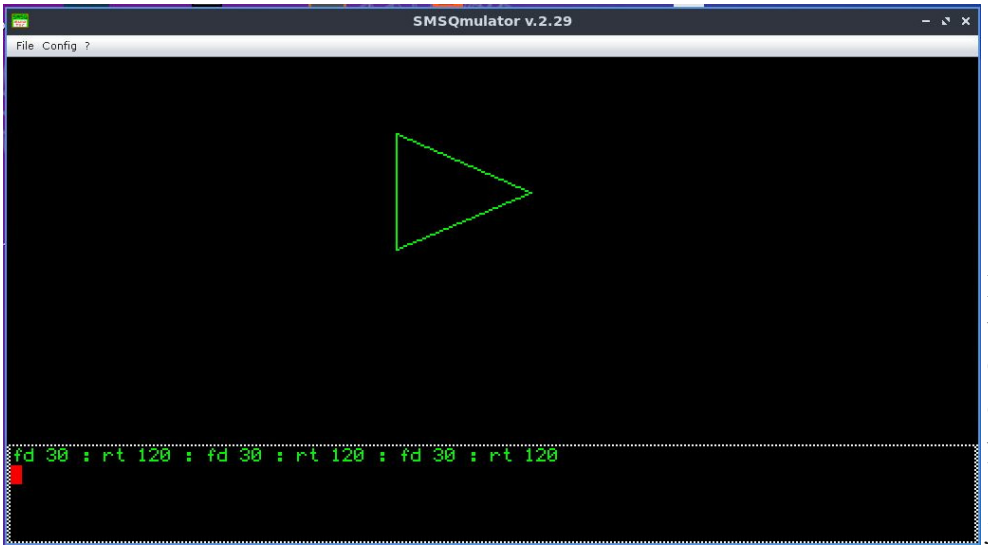
The commands are typed into the SuperBASIC command line (window #0). The procedure "init" sets up the environment and "cs" clears the screen and sets the initial location for the "turtle".

Next a number of LOGO commands are entered and the "turtle" moves leaving a line behind him. To draw a box in LOGO, here are the commands:

```
   fd 20 : rt 90 : fd 20 : rt 90 : fd
20 : rt 90 : fd 20
```

This will draw a box that is 20 units per side.

A real LOGO interpreter would not need the colons between the commands, but since we are using the SuperBASIC interpreter, we do. The commands can be entered one at a time, but it is sometimes easier to see the full "program" this way.



As with LOGO, the turtle starts at 0,0 which is the center of the screen. It uses a SCALE of 100. The coordinate system is only needed when using those commands that change the X or Y values for the turtle. The turtle starts pointing up which is 0 degrees. The degrees go around in a clock wise direction, with 90 degrees in the 3 o'clock position, 180 degrees at the 6 o'clock position, and 270 degrees at the 9 o'clock position.

To draw a triangle, use the following:

```
fd 30 : rt 120 : fd 30 : rt 120 : fd
30 : rt 120
```

To make a more interesting shape, change the last angle in the triangle:

```
fd 30 : rt 120 : fd 30 : rt 120 : fd
30 : rt 140
```

Now repeat that command a number of times. Since I'm using SMSQmulator, I just hit the up arrow and the last command comes up and I hit "enter". It is also possible to put a FOR..NEXT loop around this command to repeat it a number of times.

To add the code for a square or triangle, so you don't have to type it every time, the QL Logo source code can be edited to add a procedure of square and triangle:

```
DEFine PROCedure triangle
   fd 30 :
rt 120 : fd
30 : rt
120 : fd
30 : rt 120
END DEFine
```

Doing this, the vocabulary for QL Logo will expand and your Logo "programs" will just be procedures that you call.

The procedures or commands for QL LOGO are:

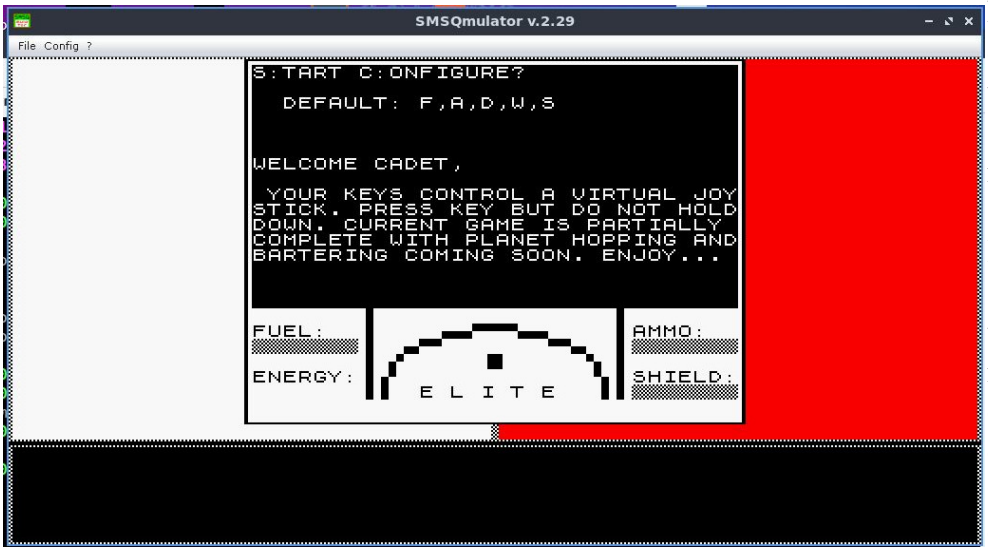
INIT - initializes the window and sets up variables.
CS - clears the screen and sets the turtle to 0,0.
FD x - Forward X units.
BK x - Backward X units.
RT x - Turn Right X degrees.
LT x - Turn Left X degrees.
PU - Pen up. No longer draws line when moving turtle.
PD - Pen down. Start drawing line when moving turtle.
SETPC x - Set Pen Color. Sets INK for drawing.
SETBG x - Set Background. Sets PAPER.
HOME - Returns the turtle back to 0,0.
CIRCLER x - Draws a circle of size X around the turtle.
CRICLEL x - Draws a circle of size X around the turtle.
SETPOS x,y - Moves turtle to X,Y and draws a line from the old position of the turtle.
DOT x,y - Draw a dot at X,Y. (does it move the turtle?)

SETX x - Sets the X position of the turtle.
SETY y - Sets the Y position of the turtle.
SETH x - Sets the heading of the turtle to X.
TURTLE - Draws a small circle at the current turtle position.

ZXsimulator

Michael Jonas has created a ZX81 basic interpreter for the QL. It runs ZX81 Basic code, just like a normal ZX81, but there is no single key entry of Basic programs. The user can type in a line of Basic, like it is done with SuperBasic.

Now, this is not a ZX81 emulator. It will not run assembly programs. It will not load a .P file to run. All programs are stored in normal text. The programs can be written with an editor and then loaded into ZXsimulator.



The whole thing is written in Digital Precision C (a version of Small-C). It loads quick and runs very fast. It comes with a few example Basic programs. It does not have a manual, so I had to look into the source code to find what the BREAK key is Ctrl-S.

When entering code into ZXsimulator, it looks just like a ZX81, with the > showing the last line. For fun I typed "edit 10" and line 10 appeared at the bottom of the screen ready for editing.

The simulator supports most BASIC commands, but does not support PEEK or POKE. It does support PLOT, inverse characters and graphics characters are supported, but only from loaded BASIC programs. Entering graphics characters is just like using zxtext2p. As of right now, it is integer only in its math routines.

The simulator comes with a couple example programs. The programs are loaded using their QL file name:

LOAD "win1_file_bas"

And programs are saved the same way with the SAVE command.

The last update was on August 9, 2021 and it is available here:

https://qlforum.co.uk/viewtopic.php?f=3&t=3193&start=130

I did a lot of Small-C back in the 90's and I did not find many others using the compiler. DP C is just an update of Small-C from the same author. ZXsimulator is 2500 lines of C code and it might be work looking at for those interested in DP C.

SSB with ZXsimulator

With ZXsimulator, programs can be typed in ZXsimulator or they can be typed into an editor and then loaded. By using an editor, the full editing powers of the editor can be used, making it easy to type in programs. There is an issue going this route in the ZXsimulator requires line numbers on all lines of BASIC.

It is possible to not use line numbers with ZX81 program when using a tool like zxtext2p, but the end result is a .P file, which ZXsimulator is not able to

handle.

For programming in BASIC, ZXsimulator is a nice tool to have, but I prefer to not use line numbers in my BASIC programs. I was thinking about what tool that could be used to convert a BASIC program without line numbers into a program with line numbers. I realized that Structured SuperBasic (SSB) is the right tool. SSB is a pre-processor for SuperBasic programs, but it does not really look at the syntax of the program. It just looks for things like labels, include files, etc. SSB would work find with ZX81 BASIC programs.

One feature of SSB is using indentation. I know that SuperBasic has no issues with indentation, so I had to test ZXsimulator. I typed in a few lines of BASIC within ZXsimulator and it handles the extra spaces just fine.

It wrote a short BASIC program that uses indentation, and saved it as test_ssb (since the SSB program assumes that the input file will end in _ssb). I then ran SSB and test_bas was the result. This loaded fine into ZXsimulator and the program ran just fine.

Here is the test program:

test_ssb:

```
print "test"
for x = 1 to 3
  for y = 1 to 3
    print x;",";y
  next y
next x
print "end test"
```

test_bas:

```
100 print "test"
110 for x = 1 to 3
120   for y = 1 to 3
130     print x;",";y
140   next y
150 next x
160 print "end test"
```

The next step was to try one of the demonstration programs that came with ZXsimulator. I took ball_bas and saved it to ball_ssb. I then edited the program, replacing GOTO numbers with labels. Once that was done, I ran it through SSB and got ball_bas. I loaded this into ZXsimulator and it ran fine.

The next step was to take a program that I had written for zxtext2p. I had to change how the labels were done. I had to change the comments. The comments in zxtext2p is a single #, where as SSB needed ##. With those minor changes done, I ran the program though SSB, then into ZXsimulator and it ran with no issues.

Since SSB supports #include files, ZX81 programs can be written in a number of files and then brought together with SSB. Since the output of SSB is regular ZX81 BASIC program, the program can be sent through zxtext2p (which will take programs with line numbers) to create a .P file.

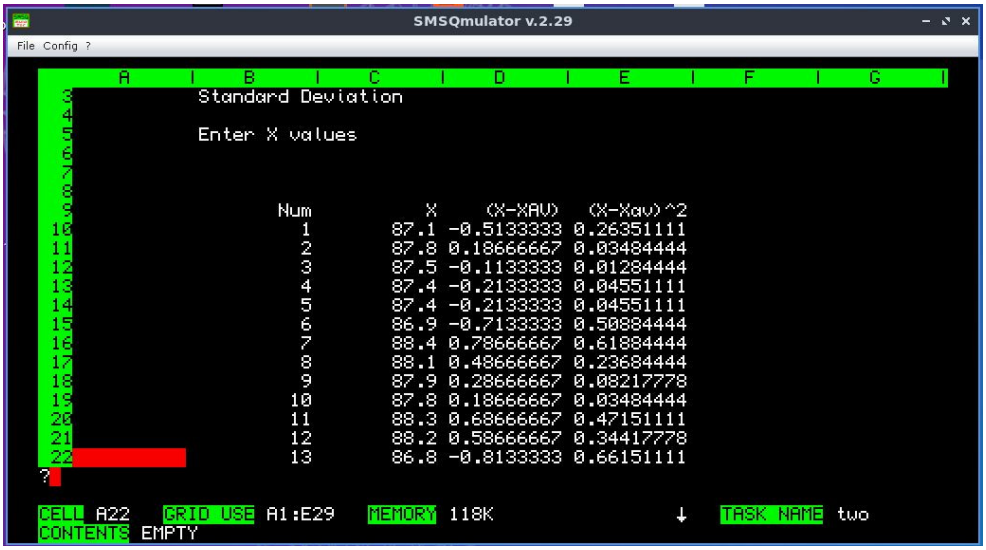
In this way, ZX81 programs can be written in SSB style, then processed to work on ZXsimulator (as text) and a ZX81 emulator (as .P). The process would be, create BASIC program as _ssb. Then use SSB to convert to _bas. From there the program can run in ZXsimulator. Then the _bas can be run through zxtext2p to create a _P file for any number of emulators.

Abacus

I've talked before about how my first job at a computer store lead me to learn Visicalc and Lotus 1-2-3. The next job, I was hired because I knew Lotus 1-2-3 and spent two years churning out Profit and Loss spreadsheets. After college, I spent a few weeks teaching a short course on spreadsheets. I got fairly good at spreadsheets.

I've also probably mentioned that I found the computational side of computers to be interesting. I did not buy my first computer to play games, but to

learn to program and do some computation.



A while back these two interested came together as I found a couple of books at local thrift stores. These stores were fairly close to a local college, so I'm guessing that some students passed their books to the thrift store.

The first book I found was "Excel Statistics: A Quick Guide" by Neil Salkind. The second book I found was "Spreadsheet tools for Engineers Using Excel" by Byron Gottfried. While tinkering with some Astronomy stuff a while back I also found "Pratical Astronomy with your Calculator or Spreadsheet" by Peter Duffet-Smith.

All three of these were interesting, but they used the more advanced features of Excel. I tried some of the examples with Ababus, but quite a number were beyond what Abacus could do.

I did some trolling on the Internet and found the book "1-2-3 for Scientists & Engineers" by William Orvis. Lotus 1-2-3 was more like Abacus than Excel, so I thought it would be a closer fit for Abacus. A number of the examples would work, but there were a few functions in Lotus 1-2-3 that Abacus did not have.

The same publishing company as the last book also published an earlier and similar book, "VisiCalc for Science & Engineering" by Trost and Pomernacki. Since Visicalc came out way before Abacus and ran in 64K, I wanted to see how those examples would translate to Abacus.

Looking at Visicalc, it has 31 functions and Abacus has 38 functions. The problem is that not all functions overlap between the two spreadsheets. Visicalc is accurate to 12 digits and Abacus is accurate to 14 digits.

Looking in the book, I found a few spreadsheets to try. The book gave step by step instructions on how to enter the spreadsheet. Between these instructions and the image of the spreadsheet, it just a few minutes to translate to Abacus.

Standard Deviation (p. 14)

This is a fairly simple example of doing standard deviation and was very simple to convert to Abacus.

Earth Satellite Orbit (p. 133)

This was a little more involved, but knowing how Abacus worked it was pretty straight forward to convert. This spreadsheet was to "determine the

height and velocity of an earth satellite having s synchronous orbit."

Simple Pendulum (p. 141)

This spreadsheet "allows you to determine the oscillation period of a simple pendulum given its length and its length given the period of oscillation." This one took a little more work as I had to convert the functions.

I found it interesting that most of the examples provided the mathematical formula for determining the result. It showed how to convert a formula to a spreadsheet syntax.

In looking into spreadsheets, I did fine some discussion of spreadsheets as a programming language. If a BASIC program can be written for determining when Easter is in a certain year, and the same algorithm can be put into a spreadsheet, I can see how it could be classified as programming.

Namecheck_pl

A while back there was some discussion on a utility that would check a SuperBASIC program and see if any variable names are the same as QL keywords. Not everyone runs with a number of tookits, so a programmer might use a variable name that is used in a toolkit and not realize it until others try out the program.

I thought a program like this could be written in a scripting language that might allow it to be done with fewer lines of code than using SuperBASIC. I picked Perl as the language to try it in.

The version of Perl ported to the QL is Perl4. What I have installed on my Linux systems is Perl5. Luckily Perl5 will run Perl4 syntax. I did the initial work on Linux. I first wrote a script that would extract assignment states and get variables from that. Then it dawned on me that Function and Procedures would also need to be checked. That was more involved and I took the brute force method.

I created a test file to run the program against and eventually I was able to get the ouput I wanted.

I also had to create a list of keywords from toolkits. I redirected EXTRAS to a file and used that as I start. I made sure to remove any duplicate keywords. If there are keywords missing from the list, they can easily be added.

When I tested the program on the QL, I found that it did not work. After some research, I found that Perl4 on the QL does not like calling subroutines with a "do sub" but uses the other syntax, "&sub". So, a few minor changes were made.

The program is run like this on the QL:

exec perl;"namecheck_pl < test2_ssb"

where test2_ssb is the source code to check. The script expects to find keywords_txt in the current directory. It also creates a var_txt to store the variables that it has extracted from the source code.

Once the second part of the program is finished, a list of variables that match keywords will be printed. If there are no results, the screen will be blank except for a "hit any key" message.